CMPSC-265
Data Structures and Algorithms

Zaihan Yang
zyang13@suffolk.edu

Department of Math and Computer Science
Suffolk University

Fall 2019

Notice

- HW6 posted. Will be due on this Sunday midnight.
- Quiz 2 on this Wednesday
- Midterm_Exam1 will be held on:
  - Time: Oct 23rd (Wednesday)
  - Location: the same classroom
  - Topics:
    - The 1st week to this week's topics

Recap

- Double-ended singly linked list
- Implementing Stack using single-ended singly linked list
- Implementing Queue using double-ended singly linked list
- Doubly linked list

Learning Topics

- Abstract data types and Interface
- Sorted linked list
- Recursion

Abstract Data Types

- A class is a data type
- Abstract data type is a class without regard to its implementation
    - What Vs How
    - Users not only don’t know how the methods work, they also don’t know what structure is used to store the data
- Example: Stack, Queue, Priority Queue
    - User knows about push and pop, but doesn’t need to know how push and pop work, or even whether data is stored in array or Linked List, or, …

Interface

- Abstract data type specification, *interface*, is what the class user sees
  - List of public methods
  - No implementation

```
public interface Stack{
        public boolean isEmpty();
        public void push(Object item);
        public Object pop();
        public Object peek();
        public void display();
        public int size();
}
```

- Implementation is delegated to classes

Interface

- An interface can be implemented by different classes

```
public class ArrayStack implements Stack
{//implement using arrays}

public class ListStack implements Stack
{//implement using Linked List}
```

- It simplifies the design

- You can change the implementation without affecting the user’s code

Sorted Linked List

- Insertion/Deletion faster than array, no need to shift.
  - Still O(n) for comparisons

- How about binary search on a sorted linked list?
  - Not effective, O(n)

Insertion in a Sorted Linked List

```
public void insert(int item)
 {
        Link newLink = new Link(item);
        Link previous = null;
        Link current = first;

        while (current!=null && current.data<item) {
        // traverse until end of list, if current=null or spot is found
                previous = current;
                current=current.next;
        }
        if (previous==null) {// check if at the beginning of the list
                first = newLink ;
        }
        else {
                previous.next = newLink ;
        }
         newLink.next = current;
}
```

Sort using a Linked List

- Given an unsorted array, we can take items one by one and insert them into a sorted linked list. It will keep them sorted, and we can move items back to the array at the end.

- Is it better than insertion sort in an array?
  - Still O(n^2) for comparisons
  - More efficient in terms of shifting/moving items
  - Needs additional memory

Recursion

- Iterative approach
  - Use loops to repeat a task, “while”, “for”, ...

- Recursive approach
  - A method calls itself
  - Each time with different parameters
  - Until we reach a trivial base case

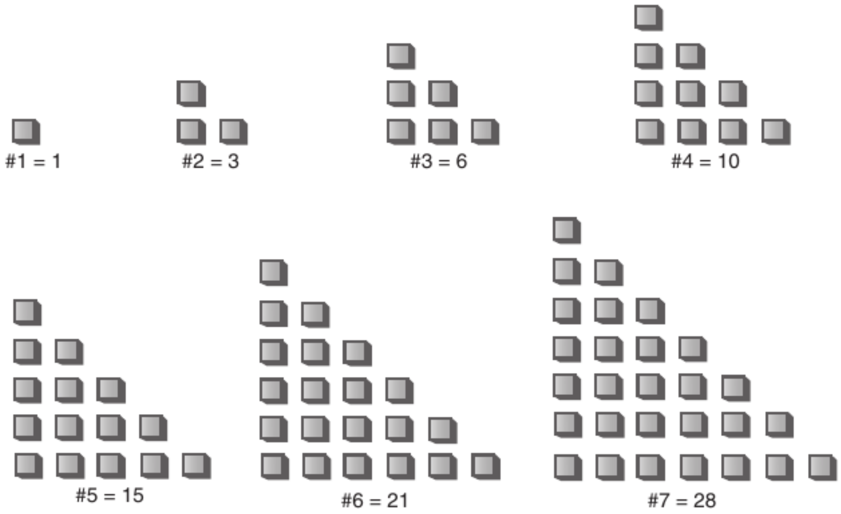
Several Recursion Problems

- Triangular numbers
- Factorial numbers
- Fibonacci numbers
- Binary Search
- All anagrams of a word
- Display the entire Linked list
- Towers of Hanoi

Example-Triangular Numbers

- Guess the next number: 1,3,6,10,15,…? 21

- Triangular numbers:
  - The *n*th term is obtained by adding n to the previous term

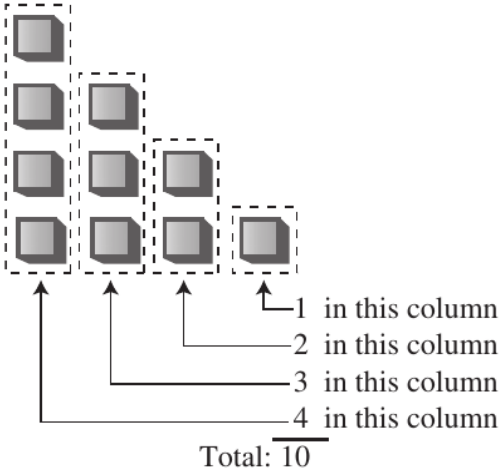


Robert Lafore. 2002. Data Structures and Algorithms in Java (2 ed.). Sams, Indianapolis, IN, USA. Page 252

Example-Triangular Numbers

- Write a method to return the *n*th term.

- Iterative approach:



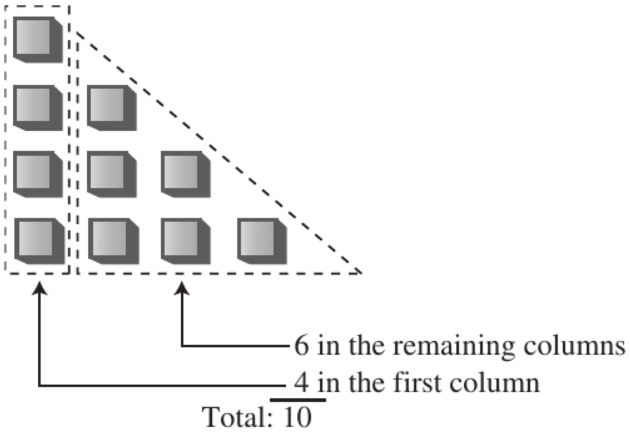
```
int triangle(int n)
{
        int total = 0;
        while(n > 0) // until n is 1
        {
                total = total + n; // add n (column height) to total
                --n; // decrement column height
        }
        return total;
}
```

Robert Lafore. 2002. Data Structures and Algorithms in Java (2 ed.). Sams, Indianapolis, IN, USA. Page 252

Example-Triangular Numbers

- Recursive approach:
  - The first(tallest) column + sum of the remaining columns

```
int triangle(int n)
{
            if(n==1)
                        return 1;
            else
                        return( n +
triangle(n-1) );
}
```



Robert Lafore. 2002. Data Structures and Algorithms in Java (2 ed.). Sams, Indianapolis, IN, USA. Page 253

How does recursion work?

- Pass 5, and print where we are

```
public static int triangle(int n)
{
        System.out.println(“Entering: n=” + n);
        if(n==1)
        {
                System.out.println(“Returning 1”);
                return 1;
        }
        else
        {
                int temp = n + triangle(n-1);
                System.out.println(“Returning “ + temp);
                return temp;
        }
}
```

Is Recursion Efficient?

- Overhead of function calls
  - Activation record should be pushed to stack
  - Contains the return address, as well

- It simplifies the problem conceptually.

- Sometimes recursion is more intuitive (e.g. trees).

Example-Factorial

- Write a method to compute n!
    - 4!=4*3*2*1
    - 3!=3*2*1
    - In general n!=n*(n-1)!
- Similar to triangular numbers, except multiplication is used instead of addition.

```
int factorial(int n)
{
        if(n==0)
                return 1;
        else
                return (n * factorial(n-1) );
}
```

Example-Recursive Binary Search

- Write a recursive version of binary search.
  - Recall the idea: divide the sorted array in half, compare the middle with the key and decide which half to continue

```
int binarySearch(int arr[], int lowerBound, int upperBound, int key)
   {
      if (upperBound >= lowerBound)
      {
         int mid = (lowerBound + upperBound ) / 2;
         if (arr[mid] == key)
           return mid;

         if (arr[mid] > key)
           return binarySearch(arr, lowerBound, mid-1, key);

         return binarySearch(arr, mid+1, upperBound, key);
      }

      return -1;
   }
```

We want to call the search method as binarySearch(arr, key)

Divide-and-Conquer

- It is an algorithm design approach
- Divide the big problem into smaller problems and solve each one separately,…
- Continue until the base case
- Recursive binary search is an example of divide-and-conquer approach

Example-Anagram

- Write a method to print all anagrams of a word.
  - Anagram of a word is a reordering of its letters

- Example: what are all the anagrams of ”abc”?
  - “abc”, “acb”, “bac”, “bca”, “cab”, “cba”

- How many anagrams with *n* letters?
  - n!

Example-Anagram

- Idea:
  - Take each letter and concatenate it with all anagrams of the remaining letters.

```
void anagrams (String s, String prefix) {
        if (s.length() == 0)
                System.out.println(prefix);
        else {
                for (int i= 0; i < s.length(); i++) {
                        String rem = s.substring(0, i) +
    s.substring(i + 1);
        anagrams(rem, prefix + s.charAt(i));
                }
        }
}
```

Example-Fibonacci Numbers

- Guess the next number: 0,1,1,2,3,5,8,…? 13

- What is the recursive relation?
  - fib(n)=fib(n-1)+fib(n-2)

- What is the base case?
  - fib(0)=0 and fib(1)=1

Example-Fibonacci Numbers

```
int fib (int n) {
        if (n<=1) // base cases
                return n;
        else
                return fib(n-1)+fib(n-2);
}
```

What is the time complexity?

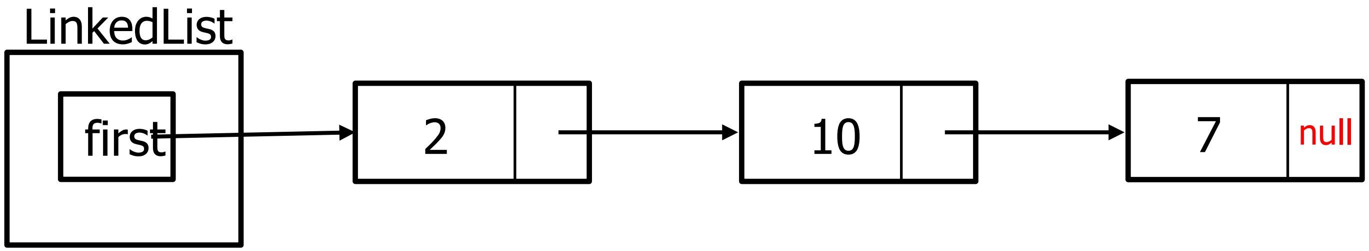
Let’s trace it for fib(5)

So, is in O(2^n)

How about space complexity?

O(n)

Example-displayList()



- Recall the iterative version:
  - Start at the beginning of the list,
  - and then Go over Links one by one using the next reference.

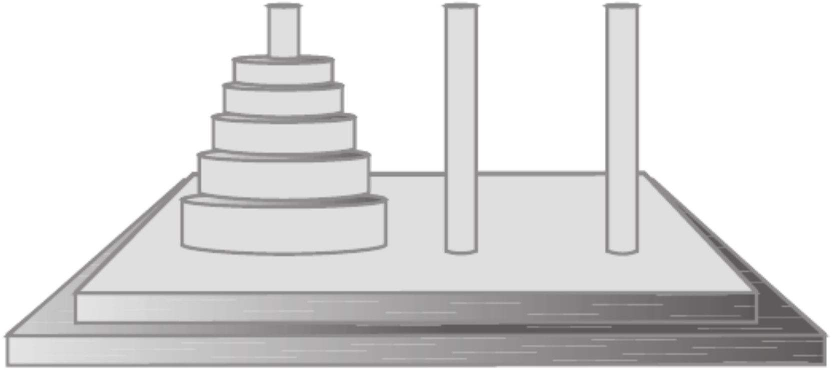
```
public void displayList()
{
        Link current = first;
        while (current!=null) // until the end of the list
        {

        System.out.println(current.data);
                current = current.next;
        }
}
```

```
public void displayList( Link current)
{
        if (current == null)
                return;
        else{
                System.out.println(current.data)
                displayList(current.next);
        }
}
```

Example-Towers of Hanoi

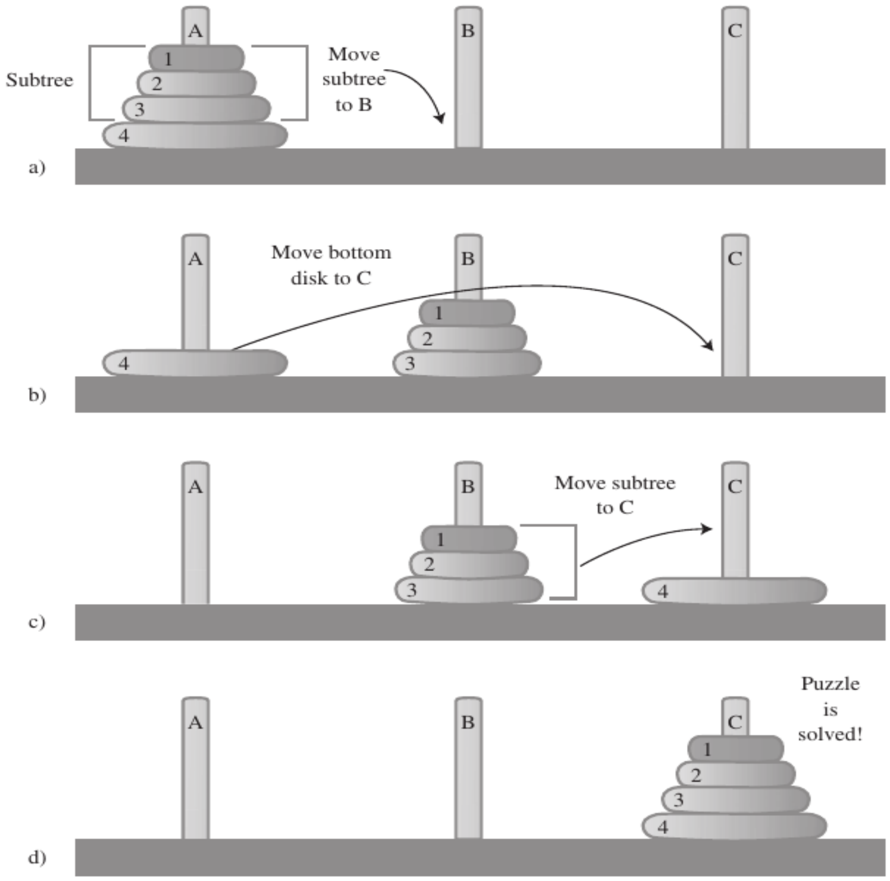
- A classic example/puzzle
  - We have 3 towers and n disks
  - We want to move disks from the first tower to the last tower
    - Move one disk at a time
    - Cannot place a disk on top of a smaller disk



Robert Lafore. 2002. Data Structures and Algorithms in Java (2 ed.). Sams, Indianapolis, IN, USA. Page 274

Example-Towers of Hanoi

- Recursion Idea
  - n=1, easy to move one
  - n=2, easy too
  - n=3?



Robert Lafore. 2002. Data Structures and Algorithms in Java (2 ed.). Sams, Indianapolis, IN, USA. Page 277

Example-Towers of Hanoi

- Recursive algorithm

```
public static void moveDisks(int n, char from, char inter, char to)
{
   if (n==1)
     System.out.print(”move 1 from ”+from+” to ”+to);
  else
    {
     moveDisks(n-1, from, to, inter);
     System.out.print(”move ”+n+” from ”+from+” to ”+to);
     moveDisks(n-1, inter, from, to);
    }
}
```